On the impact of replacing a low-speed memory bus on the Maxeler platform, using the FPGA's configuration infrastructure

Karel Heyse[1]* **, Dirk Stroobandt[1]**, Oliver Kadlcek[2]**, and Oliver Pell[2]**

[1] Ghent University, ELIS Department
Sint-Pietersnieuwstraat 41, 9000 Gent, Belgium
{Karel.Heyse,Dirk.Stroobandt}@UGent.be
[2] Maxeler Technologies Ltd.
1 Down Place, London W6 9JH, UK
{okadlcek,oliver}@Maxeler.com

Abstract. It is common for large hardware designs to have a number of registers or memories of which the contents have to be changed very seldom, e.g. only at startup. The conventional way of accessing these memories is using a low-speed memory bus. This bus uses valuable hardware resources, introduces long, global connections and contributes to routing congestion. Hence, it has an impact on the overall design even though it is only rarely used.
A Field-Programmable Gate Array (FPGA) already contains a global communication mechanism in the form of its configuration infrastructure. In this paper we evaluate the use of the configuration infrastructure as a replacement for a low-speed memory bus on the Maxeler HPC platform. We find that by removing the conventional low-speed memory bus the maximum clock frequency of some applications can be improved by 8%. Improvements by 25% and more are also attainable, but constraints of the Xilinx reconfiguration infrastructure prevent fully exploiting these benefits at the moment. We present a number of possible changes to the Xilinx reconfiguration infrastructure and tools that would solve this and make these results more widely applicable.

Keywords: FPGA, HPC, partial reconfiguration, block RAM

1 Introduction

Large hardware designs often have a number of configuration registers or memories of which the contents are changed only sporadically, e.g. at startup. The conventional way of modifying the contents of these memories is using a low-speed memory bus. Although a low-speed memory bus uses less hardware resources than a high-speed version, it will still introduce long, global connections,

* Supported by a Ph.D. grant of the Flemish Fund for Scientific Research (FWO).
** This work was partly supported by the European Commission in the context of the FP7 FASTER project (#287804).

contribute to routing congestion and affect the performance of the rest of the design.

A Field-Programmable Gate Array (FPGA) already contains a global communication mechanism in the form of its configuration infrastructure. Although it does not have the same flexibility as a custom bus, it may serve as an excellent alternative for a low-speed bus without the previously listed disadvantages.

This paper focuses on the Maxeler platform [1], a high performance computing (HPC) system consisting of a host CPU and Dataflow Engines (DFE) utilising FPGAs (Section 2). Many of the applications implemented on the Maxeler platform use *mapped memories* or small ROMs and RAMs that can be accessed locally from hardware and globally from the host CPU. Mapped memories are implemented on the FPGA using block RAM (BRAM) primitives so that local access is fast. Global access happens via a low-speed mapped memory controller connecting the host CPU to the mapped memories. In common use, this happens at most every few hundred milliseconds.

In this paper, we investigate the use of partial reconfiguration of block RAMs to replace this low-speed memory bus for global mapped memory access (Section 3). Partial reconfiguration means that the configuration infrastructure of the FPGA is used to change the configuration of a part of the FPGA while the remainder continues to operate without interruption.

Partial reconfiguration is typically used to improve the functional density of designs. This is for instance done by loading and unloading modules as needed [2, 3] or by performing dynamic circuit specialisation [4]. This makes it possible to use smaller and cheaper FPGAs to do the same amount of computation as larger FPGAs that do not use partial reconfiguration.

Partial reconfiguration of block RAMs has been proposed in [5] as a way to implement Network-on-Chips on FPGAs with reduced hardware resource cost. In this work, partial reconfiguration is used to transfer data from sender to receiver by temporarily storing the data in a sender block RAM and relocating the configuration bitstream of this block RAM to a receiver block RAM. However, no previous research has ever investigated how partial reconfiguration of block RAMs can be used to improve the routability and maximum clock frequency of applications.

Our experiments on three real-world designs, one financial application and two geoscience applications, have shown that the low-speed memory bus as currently implemented causes routability issues and limits the clock frequency in some designs. In those cases, the use of partial reconfiguration instead of this bus results in higher clock frequencies – up to 8% in our experiments – and thus better quality designs (Section 5). Improvements of 25% and more were also attained, but constraints of the Xilinx configuration infrastructure prevent us to fully exploit these benefits at the moment. In Section 6, we present a number of small improvements to the Xilinx configuration infrastructure and tools that would solve this problem and make these results more widely applicable.

2 Maxeler Platform Background

The Maxeler platform, developed by Maxeler Technologies, is a system for high performance computing consisting of a host CPU and hardware accelerators called Dataflow Engines (DFE) [1]. This section provides an overview of the Maxeler platform and toolchain. It also describes the mapped memory component and the low-speed memory bus by which they are connected.

2.1 Hardware Platform

Maxeler produces several variants of its hardware platform optimised for different computing needs. In general, the platform consists of one or more conventional host CPUs and one or more DFE coprocessors (DFEs) (Figure 1). In the system we study, the DFEs are connected to the CPUs using PCI Express and to each other directly over a custom high-bandwidth MaxRing network.

The host CPUs are used for managing the DFEs, by triggering configuration of the required bitstreams and streaming data to and from the DFEs using DMA streams, and for computations that do not need to be executed in hardware.

The MAX3 DFE used in this work has a large Xilinx Virtex 6 FPGA, called the Compute-FPGA (CFPGA), and a smaller auxiliary Xilinx Virtex 6 FPGA, called Interface-FPGA (IFPGA). This IFPGA is used for managing the PCIe communication between the host and CFPGA and for configuration of the CFPGA via its SelectMap interface. The IFPGA itself is configured from flash memory at startup, while the CFPGA is (re)configured with an application-specific bitstream provided by the host CPU every time a new application is started. The MAX3 also contains 24 GB of DRAM directly accessible from the CFPGA.

2.2 Toolchain

The Maxeler toolchain raises the implementation level for the application developer above the hardware level and thus reduces the development effort for DFE applications.

The part of an application that is accelerated in hardware is called a *kernel*. To implement a kernel on a DFE, the developer has to create a dataflow model of it. This is a description of the logic and arithmetic datapath through which streams of data flow and by which new streams of data are produced.

The description of a complete application for the Maxeler platform consists of three parts: a dataflow model of the kernel(s) in MaxJ (Maxeler's extension of the Java language), a description (also as a dataflow model in MaxJ) of the manager describing how the kernel(s) communicate with the CPU, other DFEs and memory, and the host application that is run on the CPU.

Using the Maxeler toolchain the dataflow model and manager description are compiled into an FPGA configuration bitstream for the hardware accelerator. The host code, in C or any of the other supported languages, is compiled and linked with Maxeler's SLiC and MaxelerOS run-time libraries, which enable it to communicate with the DFEs.

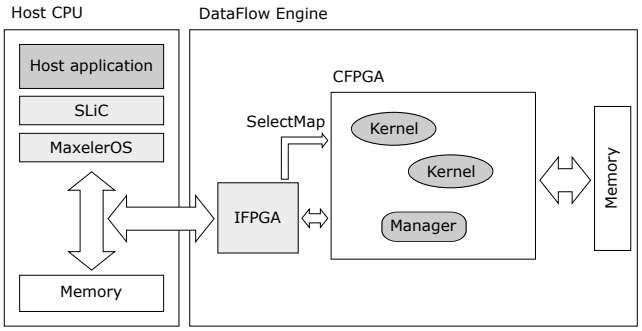

Fig. 1. Maxeler platform with one host CPU and one DFE. The user-defined parts are marked in dark grey.

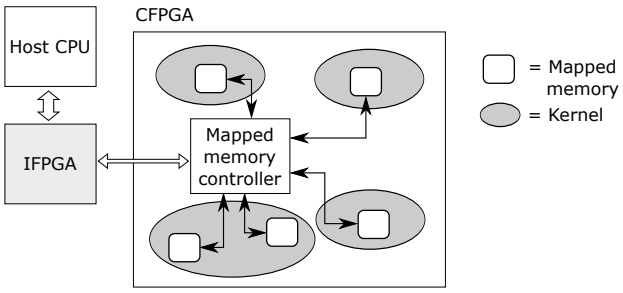

Fig. 2. Mapped memories and the mapped memory controller.

2.3 Mapped Memories

Mapped memories are small RAMs or ROMs inside kernels that are implemented using BRAMs and can be accessed with low latency from local hardware. The difference with respect to regular RAMs and ROMs is that mapped memories can also be read and written globally from the host CPU. Mapped memories are therefore often used for configuration data, like filter coefficients, that need to be changed between runs of the application but usually not during the processing of a datastream. Other use cases are 'working memory' that needs to be initialised or of which the final state needs to be retrieved. In common use cases, global access to these memories occurs at most every few hundred milliseconds.

Global access is enabled by a custom low-speed, low-overhead bus and is a lot slower than local access. This 32-bit wide, low-speed bus consists of a mapped memory controller running at 50MHz that is connected in star topology to all the mapped memories (Figure 2). The controller is connected via the IFPGA to the host CPU using Programmed Input/Output (PIO). The mapped memory controller decodes read and write commands received from the host CPU and controls the read and write signals of the different mapped memories.

Because the mapped memory controller is connected to all the mapped memories, which may be spread out over the complete CFPGA, it can become a rout-

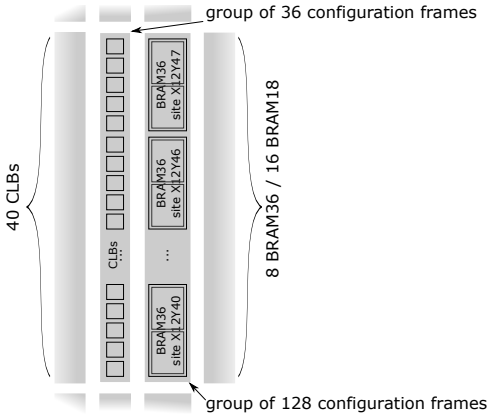

Fig. 3. Groups of configuration frames and stacks of BRAMs and CLBs.

ing bottleneck. Also the long, device spanning connections from remote mapped memories to the typically centrally located mapped memory controller can cause timing issues that are hard or impossible to resolve.

3 Implementation of Mapped Memories Using Partial Reconfiguration

The Xilinx Virtex 6, used in the MAX3, supports configuration readback and partial reconfiguration of a small portion of the FPGA while the rest of the FPGA remains operational. Because the contents of the BRAMs are also part of the FPGA's configuration, this feature can be used to read and write BRAMs.

By using partial reconfiguration of BRAMs to read and write the mapped memories, the use of the mapped memory controller can be avoided and the long, timing-sensitive connections can be removed.

In this section we will first provide more details about partial reconfiguration of BRAMs, then describe the changes made to the hardware, compilation toolchain and run-time libraries.

3.1 Partial Reconfiguration of BRAMs

Partial reconfiguration and configuration readback are done by sending special bitstreams from the host CPU to one of the FPGA's configuration interfaces: JTAG, SelectMap or ICAP (Internal Configuration Access Port) [6]. This bitstream contains the commands to read or write a specific portion of the FPGA's configuration.

The minimum granularity by which the FPGA configuration can be accessed, i.e. the smallest unit of data with its own address, is a configuration frame (2592 bits). Together, a group of frames describes the configuration of a section of the

FPGA's resources (Figure 3) [7]. A group of 36 frames, for example, describes the configuration of a partial column of CLBs. A group of 128 frames describes the contents of a partial column or stack of 8 BRAM36s or 16 BRAM18s, since each BRAM36 can also be split into two BRAM18s. The BRAMs that are described in the same group of configuration frames, and therefore belong to the same stack, have the same X coordinate and the same Y coordinate after integer division by 8 for a BRAM36 or by 16 for a BRAM18.

The configuration infrastructure of the BRAMs is implemented in hardware by sharing read/write ports with the FPGA fabric. Because of this it is unsafe to use a BRAM in the FPGA fabric – keeping the clock running and enable signal active – while readback of its configuration frames is taking place. As a result, if configuration readback of one BRAM is performed, the complete BRAM stack containing it must be halted.

According to [8, 5] it is possible to mask specific BRAM36s (but not BRAM18s) during reconfiguration so that their configuration is not overwritten. Our experiments seem to imply that if a BRAM is masked in this way, and only then, it is safe to use it in the FPGA fabric while reconfiguration of its frames is happening. Unfortunately, no similar masking function is available for configuration readback.

3.2 Configuration Interface

To enable communication between the host CPU and the FPGA's configuration port the *ICAPStreamingBlock* is implemented on the CFPGA. This block, operating at 60MHz, connects the ICAP to two PCI Express streams, one in each direction. The ICAPStreamingBlock receives a combined bitstream and command stream from the host CPU. The bitstream is passed to the data port of the ICAP and the command stream sets the ICAP control signals and tells the ICAPStreamingBlock whether data is expected on the output port of the ICAP. This output data is then streamed back to the host CPU.

Alternatively, this functionality could be implemented on the IFPGA, which is already connected to the SelectMap configuration interface. Previous work has shown that the IFPGA can be used to partially reconfigure the CFPGA[9], but it is currently only able to (re)configure the FPGA and not to perform configuration readback. Adding readback support is straightforward engineering but we have not implemented this as part of this work since it would require significant development time and it is possible to assess the impact on the CFPGA without this feature being operational.

3.3 Changes to the Compilation Toolchain

We adapt the compilation toolchain so that partial reconfiguration can be used to perform global access of mapped memories. The underlying implementation of the global access method is transparent to the developer of the DFE and it is simple to switch between the original and new access method using the *UseMicroreconfigMem* option in the MaxJ description of the kernel (Figure 4).

```
optimization.pushUseMicroreconfigMem(true);
Memory<DFEVar> ram = mem.alloc(type, size);
ram.mapToCPU("mapped_ram_name");
optimization.popUseMicroreconfigMem();
DFEVar x = ram.read(addr);
```

Fig. 4. Instantiating a mapped memory with global access implemented using partial reconfiguration.

The compilation toolchain disconnects these mapped memories from the mapped memory controller and extracts the placement and port width of the BRAMs so that this information can be used by the run-time libraries.

Instead of Xilinx Coregen, we have implemented our own mechanism for composing large memories from BRAM primitives. This is done because the exact way Coregen combines BRAMs to form larger memories is not documented and this information is required to map the contents of the mapped memories to the correct parts of the FPGA's configuration. This information is also exported for use by the run-time libraries.

3.4 Changes to the Run-Time Libraries

Instead of passing the read and write commands from the host code to the mapped memory controller, the MaxelerOS run-time library must now interpret these commands and translate them into the necessary reconfiguration and readback actions.

First, it must find out in which physical BRAM(s) a memory element is stored (Figure 5) and calculate the corresponding configuration frame addresses. This is done using the information that was exported by the compilation toolchain.

To perform a read operation on a BRAM, a special readback bitstream is sent to the FPGA's configuration interface causing the contents of the BRAM's configuration to be sent back. The mapping of the contents of the BRAM to the configuration bits is then reversed to extract the requested element. If a memory element is spread out over multiple BRAMs this procedure is repeated for each BRAM.

For a write operation we need to take into account that it is not possible to update a single memory element because a reconfiguration always updates at least a full BRAM36 (or 2 BRAM18s). Therefore, to correctly perform a write operation the relevant portion of the current configuration of the BRAM must be obtained, modified locally on the host CPU and then written back to the FPGA.

Because the contents of a mapped ROM will only be written from the host CPU and not locally on the FPGA, it is sufficient to keep a local mirror of the BRAMs' configurations on the host CPU to know their current configuration at all times. This local mirror must be updated in sync with the configuration of the FPGA.

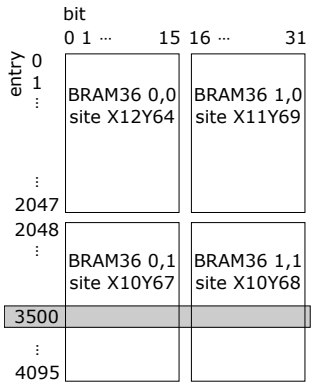

Fig. 5. Large block memory combining 4 BRAM36s (port width: 16 bits). The first 16 bits of element 3500 are stored in element 1452 of the BRAM on site X10Y67, the last 16 bits in element 1452 of the BRAM on site X10Y68.

For a mapped RAM the only way to know the current configuration of the BRAM is to perform a configuration readback. It is important that the contents of the BRAM do not change between the readback and reconfiguration operations because such an update would be lost.

Global access to mapped memories often happens in batches; a large part of a mapped memory is read or written at the same time. To improve efficiency, multiple accesses to the same mapped memory that affect the same configuration frames are grouped together so that the number of configuration readback and reconfiguration operations is reduced.

4 Challenges: Partial Reconfiguration Constraints

In this section we discuss how the constraints imposed by the configuration infrastructure, introduced in Section 3.1, affect applications on the Maxeler platform. We recall that during configuration readback the complete stack of BRAMs that is being read cannot be used and during reconfiguration only the BRAMs that are masked can be used.

Write operations to ROM mapped memories, which do not require a configuration readback, can therefore be performed safely as long as only BRAM36 primitives are used for their implementation and the other BRAMs are masked during reconfiguration.

In contrast to write operations on ROMs, read and read-modify-write operations on RAM mapped memories require that the complete stack of BRAM primitives containing the BRAM to be accessed is halted. For applications on the Maxeler platform it may be safely assumed that BRAMs from the same kernel as the mapped memory will be halted when the mapped memory is being accessed, but no assumptions about other BRAMs can be made. This will be a common case in many scenarios where part of a chip must continue running while another

part is paused for reconfiguration. Placement constraints must therefore be used to prevent other BRAMs being placed in the read back stacks. In general, (automatic) floorplanning could be used to achieve this, previous experiments by Maxeler have shown that overly aggressive floorplanning has significant detrimental effects on maximum clock frequency, so we do not believe this to be a feasible option.

An alternative method using multiple iterations of Place & Route (P&R) has also been tested. In this method each incremental P&R run adds constraints (LOC and PROHIBIT) based on previous runs until the location constraints for reconfigurable BRAMs are met. This also had an unacceptable impact on the clock speed of anything but the smallest applications. This is not entirely unexpected because, for the same reason as the problems with floorplanning in general, large numbers of constraints are known to make it harder to find a feasible P&R solution.

While ROM mapped memories using partial reconfiguration can be used on the Maxeler platform, no feasible solution has been found for RAMs for realistically sized applications. In Section 6 we propose a number of changes to the configuration infrastructure and Xilinx tools that would make it possible to use RAM mapped memories on the Maxeler platform and other platforms with similar constraints. Note that on certain other platforms the location constraints may be resolved using floorplanning or circumvented by stopping all BRAMs.

5 Evaluation: Clock frequency benefits

5.1 Evaluated Applications

We have evaluated the proposed method on one financial application related to price prediction of financial derivatives and two geoscience applications related to detecting underground oil and gas reserves based on acoustic reflections. We will call them FINAN, GEO1 and GEO2. The evaluated designs are real-world applications developed by Maxeler and its customers.

Table 1. Description of the applications

	Logic	DSP	BRAM	Mapped memories
FINAN	87%	81%	51%	36 x 624 elem. x 32 bit
GEO1	69%	22%	50%	27 x 50 elem. x 23 bit
GEO2	70%	59%	70%	6 x 2002 elem. x 17 bit + 3 x 1024 elem. x 18 bit + 33 x $\leq$128 elem. x 18 bit

Table 1 contains a summary of the resource utilisation and mapped memories of each application. All three applications use only ROMs and these have values which are loaded at the start of each compute job (once every few minutes to hours).

All applications additionally contain 2 small mapped memories that cannot be implemented with partial reconfiguration so the mapped memory controller has to be retained. Even though the mapped memory controller cannot be removed, the routing bottleneck is resolved by disconnecting the majority of the mapped memories from the controller.

5.2 Maximum Clock Frequency

Table 2. Maximum clock frequency (MHz). Items with * are currently not functional

		Partial reconfiguration	
	Conventional	ICAP	SelectMap*
GEO1 (ROM)	100	90 (0.90)	100 (1)
GEO1 (RAM)*	80	90 (1.13)	100 (1.25)
GEO2 (ROM)	130	140 (1.08)	140 (1.08)
GEO2 (RAM)*	< 80	130 (> 1.63)	130 (> 1.63)
FINAN (ROM)	180	180 (1)	180 (1)
FINAN (RAM)*	180	180 (1)	180 (1)

Table 2 contains the maximum clock frequencies for the three applications in different configurations. The "Conventional" column contains the maximum clock frequencies using the conventional implementation of mapped memories, the "ICAP" and "SelectMap" columns show the maximum frequencies using partial reconfiguration. For the values in the "SelectMap" column, the IFPGA is used to perform partial reconfiguration via the external SelectMap configuration port of the CFPGA (Section 3.2) instead of the "ICAP" interface and ICAPStreamingBlock on the CFPGA itself. This is currently not functional but it is straightforward to see how it would operate.

In the ROM cases, the use of partial reconfiguration improves the maximum clock frequency of GEO2 by about 8% and leaves the other applications unchanged (SelectMap). The introduction of the ICAPStreamingBlock, however, causes a reduction of the maximum clock frequency of about 10% for GEO1. The reason for this is that the ICAP and DDR memory controller both are constrained to the same, congested area of the FPGA. The use of the SelectMap configuration interface would alleviate this problem.

In the RAM cases we assume, for the sake of the experiment, that the mapped memories are RAMs instead of ROMs and that the placement constraints for RAMs (Section 4) do not exist. In these applications the primary use of RAMs is for providing debug visibility, however other applications would also use RAMs in production mode. The conventional implementation of mapped RAMs causes extra routability issues and increases the advantage of the proposed method. For GEO1 we now see a frequency increase of 13-25% and for GEO2 we find that while it was impossible to meet the minimal timing constraints using the conventional method, a clock frequency of 130MHz can now be attained.

Because in this experiment we are ignoring the placement constraints for partially reconfigurable RAMs, the result of this experiment can currently not be used in practice, however this experiment illustrates the potential benefits of this approach if modest improvements were made to the partial reconfiguration mechanism of the FPGA (as we discuss in Section 6). We successfully ran all the applications in ROM mode.

The relatively high clock frequency of the FINAN application is not affected by the method used to implement the mapped memories. This shows that not in every application the mapped memory controller is a routing bottleneck. There is currently insufficient data to make conclusions about which type of applications will benefit the most from the proposed implementation method. Our preliminary findings, however, show that applications that require a larger effort to solve the original P&R problem with low-speed bus generally benefit more.

Experiments have shown that read and write speeds of mapped memories implemented using partial reconfiguration (read: 2-14 Mbit/s, write: 1-10 Mbit/s) remain in the same order of magnitude as with the conventional implementation (read: 2-4 Mbit/s, write: 7-14 Mbit/s). The difference is of small importance because the read and write times are very small compared to the total execution time.

The solutions using partial reconfiguration used at most 1% more logic resources and 2% more BRAM resources than the conventional implementations. The additional resources were needed for the ICAPStreamingBlock and in some cases because BRAM18 primitives had to be replaced by BRAM36 primitives. We believe that this hardware cost is acceptable for an 8% speed improvement.

6 Recommendations

In this section we make a number of recommendations for possible changes to the FGPA configuration infrastructure and P&R tools that would make the proposed method more widely applicable.

Configuration readback of a BRAM can currently only be done when all the BRAMs in the stack containing it are halted or unused (Section 4). This can be achieved by locking the BRAM to a specific location and prohibiting the other BRAMs in the same stack from being used, by shutting down all the BRAMs on the FPGA during configuration readback or, if floorplanning is used, by turning off all the BRAMs in the region containing the BRAM to be read. For many applications, including the ones implemented on the Maxeler platform, these are infeasible solutions. We present three possible ways to solve this problem.

A first possible solution is to change the Placement & Routing algorithm so that it does not place the BRAMs of which we want to read back the configuration in the same stacks as BRAMs that we do not wish to halt during reconfiguration. As has been shown, a work-around method using multiple passes of the existing P&R tools is insufficient, but an integrated algorithm might achieve better results. We present the option because it is the only solution that does not require changes to the physical FPGA architecture.

A second option would be read masking – a straightforward extension of the existing write masking. This would allow independent configuration readback of BRAMs even if they are located in the same stack. Because the configuration infrastructure already supports write masking, we believe that it would be feasible to implement the same for readback operations in future FPGA architectures.

Alternatively, the bits of the BRAMs in the configuration frames can be rearranged so that data of only one BRAM is stored in each frame, as opposed to the current situation where content from multiple BRAMs is striped across multiple frames.

Finally, we recommend to provide more possible locations to place the ICAP port than the currently available two options, which are located close together. This would help to avoid the routing congestion caused by logic modules that need to be constrained to the same region.

7 Conclusion

We have proposed a method to access block memories on FPGAs using partial reconfiguration and configuration readback. The use of this method removes the need for the low-speed memory bus that is conventionally used for this purpose. Using three real-world applications, we have shown that a maximum clock frequency improvement of up to 8% is possible because of this. The proposed method can currently be applied to all applications of which the block memories only need to be written. We have proposed a number of small improvements to the Xilinx configuration infrastructure and tools that would make it possible to achieve clock speed improvements of 25% and more when block memories need to be read as well as written.

References

1. Pell, O., Mencer, O., Tsoi, K.H., Luk, W.: Maximum Performance Computing with Dataflow Engines. High-Performance Computing Using FPGAs (2013) 747–774
2. Beckhoff, C., Koch, D., Torresen, J.: GoAhead: A Partial Reconfiguration Framework. In: Field-Programmable Custom Computing Machines (FCCM), IEEE 20th Annual International Symposium on. (2012) 37–44
3. Xilinx: Partial Reconfiguration User Guide. (2010)
4. Bruneel, K., Heirman, W., Stroobandt, D.: Dynamic Data Folding with Parameterizable FPGA Configurations. ACM Transactions on Design Automation of Electronic Systems (TODAES) **16**(4) (2011) 43:1–43:29
5. Shelburne, M., Patterson, C., Athanas, P., Jones, M., Martin, B., Fong, R.: MetaWire: Using FPGA Configuration Circuitry to Emulate a Network-on-Chip. In: Field Programmable Logic and Applications, International Conf. on. (2008) 257–262
6. Xilinx: Virtex-6 FPGA Configuration User Guide. (2012)
7. Xilinx: Virtex-5 FPGA Configuration User Guide. (2012)
8. Xilinx: XAPP290 Difference-Based Partial Reconfiguration. (2007) 1–11
9. Cattaneo, R., Pilato, C., Mastinu, M., Kadlcek, O., Pell, O., Santambrogio, M.: Runtime Adaptation on Dataflow HPC Platforms. In: Adaptive Hardware and Systems (AHS), NASA/ESA Conference on. (2013) 84–91